中间代码生成

2020年7月3日 18:27

1. 类型表达式(Type Expressions)
    a. 基本类型是类型表达式
    b. 可以为类型表达式命名，类型名也是类型表达式
    c. 将类型构造符(type constructor)作用于类型表达式可以构成新的类型表达式
        i. 如array是数组类型构造符，若T是类型表达式，则array(I, T)是类型表达式(I是一个整数)
            1) 如int[3]类型的类型表达式array(3, int)
            2) 如int[2][3]类型的类型表达式array(2, array(3,int))
        ii. 如指针构造符pointer，若T是类型表达式，则pointer (T) 是类型表达式，它表示一个指针类型
        iii. 笛卡尔乘积构造符x，若T1和T2是类型表达式，则笛卡尔乘积T1xT2 是类型表达式
        iv. 函数构造符→，若T1、T2、…、Tn 和R是类型表达式，则T1xT2x…xTn→R是类型表达式
        v. 记录构造符record，若有标识符N1、N2、…、Nn 与类型表达式T1 、T2、…、Tn ，则 record((N1xT1) x (N2xT2) x…x (NnxTn))是一个类型表达式
2. 声明语句的翻译
    a. 声明语句语义分析的主要任务：收集标识符的类型等属性信息，并为每一个名字分配一个相对地址
    b. 局部变量的存储分配
        i. 从类型表达式可以知道该类型在运行时刻所需的存储单元数量称为类型的宽度(width)
        ii. 在编译时刻，可以使用类型的宽度为每一个名字分配一个相对地址
    c. 名字的类型和相对地址信息保存在相应的符号表记录中
        i. 形如enter(name, type, offset):在符号表中为名字name创建记录,将name的类型设置为type,相对地址设置为offset
3. 简单赋值语句的翻译
    a. 赋值语句翻译的主要任务：生成对表达式求值的三地址码
        i. lookup(name)：查询符号表，返回name对应的记录
        ii. gen(code)：生成三地址指令code
        iii. newtemp( )：生成一个新的临时变量t， 返回t的地址
    b. 增量翻译(Incremental Translation)
        i. 不需要设置code属性，直接在生成好的三地址码后增加新的三地址指令
        ii. 在增量方法中，gen( )不仅要构造出一个新的三地址指令，还要将它添加到至今为止已生成的指令序列之后
4. 数组引用的翻译
    a. 将数组引用翻译成三地址码时要解决的主要问题是：确定数组元素的存放地址，也就是数组元素的寻址
        i. 一维数组：假设每个数组元素的宽度是w，则数组元素a[i]的相对地址是：base+i*w。其中，base是数组的基地址，i*w是偏移地址
        ii. 二维数组：假设一行的宽度是w1，同一行中每个数组元素的宽度是w2，则

数组元素a[i1] [i2]的相对地址是： base+i1*w1+i2*w2

iii. k维数组：数组元素a[i1] [i2] ...[ik]的相对地址是： base+i1*w1+i2*w2+...+ik*wk。其中，w1→a[i1]的宽度 w2→a[i1] [i2]的宽度 ... wk→a[i1] [i2]...[ik]的宽度

5. 控制流语句的SDT
   a. if 布尔表达式 被翻译成由跳转指令构成的跳转代码，其继承属性：
      i. S.next：一个地址，该地址中存放了紧跟在S代码之后的指令(S的后继指令)的标号
      ii. B.true：一个地址，该地址中存放了当B为真时控制流转向的指令的标号
      iii. B.false：一个地址，该地址中存放了当B为假时控制流转向的指令的标号
      iv. 这些指令的标号标识了三地址指令
   b. 控制流语句的SDT
      i. newlabel( ): 生成一个用于存放标号的新的临时变量L，返回变量地址。常在产生式中新标识符前使用
      ii. label(L): 将下一条三地址指令的标号赋给L。常在产生式新标识符后使用
6. 布尔表达式的SDT
   a. 在跳转代码中，逻辑运算符&&、|| 和! 被翻译成跳转指令。运算符本身不出现在代码中，布尔表达式的值是通过代码序列中的位置来表示的
7. SDT的通用实现方法
   a. 首先建立一棵语法分析树，然后按照从左到右的深度优先顺序来执行这些动作
8. 布尔表达式的回填(Backpatching)
   a. 基本思想：
      i. 生成一个跳转指令时，暂时不指定该跳转指令的目标标号，将这样的指令都被放入由跳转指令组成的列表中
      ii. 同一个列表中的所有跳转指令具有相同的目标标号
      iii. 等到能够确定该目标标号时，才去填充这些指令的目标标号
   b. 布尔表达式的指令列表
      i. 综合属性B.truelist：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是当B为真时控制流应该转向的指令的标号
      ii. 综合属性B.falselist：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是当B为假时控制流应该转向的指令的标号
   c. 处理列表的函数
      i. makelist(i)：创建一个只包含i的列表，i是跳转指令的标号，函数返回指向新创建的列表的指针
      ii. merge(p1, p2)：将p1和p2指向的列表进行合并，返回指向合并后的 列表的指针。常配合在and运算符和or运算符后新增的M→ε使用（该产生式用于记录p2的标号）
      iii. backpatch(p, i)：将i 作为目标标号插入到p所指列表中的各指令中
   d. 例
9. 控制流语句的回填

a. 综合属性S.next1ist：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是按照运行顺序紧跟在S代码之后的指令的标号

10. switch语句的翻译
    a. 方法A：在case V1后补一句if t != V1 goto L1，然后再case V2后补一个label L1，再不断重复
    b. 方法B：给每个case都建一个label，将标号和case值存进map，将跳转判断指令集中在一起
    c. 有时会为这种if else增加一个冗余的case指令，含义是相同的，只是为了方便最终的代码生成器探测到此处可以特殊处理
11. 过程调用语句的翻译
    a. 用一个队列存放实参列表，计算出每个实参表达式的值后才调用

```
n=0;
for q中的每个t do {
        gen( ‘param’  t );
        n = n+1;  }
gen( ‘call’  id.addr  ‘,’  n);
```